Commodore-Software

AUSTRO-COMP

-COMMODORE BASIC 4.0 Compiler-

commodore COMPUTER

Inhaltsverzeichnis Seite

1.	Einleitung	3
1.1	Programmschutz	4
2.	Copyright	5
3.	Lizenz	5
4.	Allgemeines	6
5.	Kompilieren	8
5.1	Pass 1	9
5.2	Pass 2	11
6.	Programmarten	
6.1	Ein Programm kompilieren nach Drive 1	12
6.2	Alle Programme von Drive 0 nach Drive 1	14
6.3	Alle Programme von Drive 0 nach Drive 1 (erstes Programm mit Comp-Maschinenprogramm)	16
6.4	Alle Programme von Drive 0 nach Drive 1 mit Overlay	17

Inhaltsverzeichnis Seite

7.	Spezialanweisungen	19
8.	Arithmetik und Funktionen	21
9.	Arrays	23
10.	Stacktiefe	23
11.	Fehlermeldungen	25
12.	Maschinenprogramme	27
13.	Sonderanweisungen	29
14.	AUSTRO-COMP Daten	30
15.	Stichwortverzeichnis	31

1. Einleitung

AUSTRO-COMP ist ein BASIC-Compiler für Commodore-BASIC bis 4.0.
Es ist mit den BASIC-Interpretern der CBM-Systeme 3032, 4032 und 8032 voll kompatibel.
Insbesondere die Pointer und Variablen werden in gleicher Art geführt. wie die von BASIC. Deshalb ist der AUSTRO-COMP auch für Maschinenunterprogramme kompatibel.

Besonderheiten des AUSTRO-COMP

- Kompilieren von Warm-Overlay-Programmpaketen

- automatisches Durchkompilieren von Programmpaketen ist mit dem AUSTRO-COMP möglich

- Verarbeitung von BASIC-Extensions erfolgt BASIC-kompatibel. Deshalb können Programmierhilfen und -Zusätze weiterverwendet werden

- Übernahme von Maschinenprogrammen in kompilierte Programme

Der AUSTRO-COMP ist ein 2-Pass-Compiler. lediglich Overlays werden in 3 Passes kompiliert. Das kompilierte Programmm besteht aus RUN-TIME Routine diese ist 4 k Bytes groß, dem P-Code. Er beinhaltet ca. 50% der Länge der BASIC-Source und Variablen. Die Variablenbeginn-Pointer werden auf den Beginn der Variablen gesetzt. Programme ab einer Länge von ca. 9 k Bytes werden damit verkürzt. wodurch sich der Variablenspeicher erhöht.

Eine genaue Geschwindigkeitsangabe für kompilierte Programme ist nur schwer möglich, weil die Beschleunigung von Befehl zu Befehl variiert. Bei Nutzung der Floppy-Disk-Einheiten ergibt sich eine durchschnittliche Vervierfachung gegenüber der Verarbeitungszeit eine BASIC-Programmes.

1.1 Programmschutz

Mit AUSTRO-COMP können kompilierte Programme mit einem AUSTRO-COMP-Dongle geschützt werden. Dadurch kann Verletzungen des Urheberrechts, sowie dem unerlaubten Weiterverkauf der Programme vorgebeugt werden.

2. Copyright

Das Copyright ist Eigentum der Commodore GmbH., Frankfurt/Main

3. Lizenz

AUSTRO-COMP wird unter der Bedingung des Nutzungsrechts verkauft. Dies bedeutet, daß diese Software nur auf der vom Käufer angegebenen Anlage eingesetzt werden darf.
Das Kopieren der Software, des Handbuchs oder anderer Literatur im Zusammenhang mit AUSTRO-COMP ist nicht gestattet. Der Käufer verpflichtet sich, dem Personenkreis, der mit AUSTRO-COMP arbeitet von diesen Bedingungen zu unterrichten.

4. Allgemeines

AUSTRO-COMP ist ein BASIC-COMPILER, der voll kompatibel mit COMMODORE Basic 4.0 ist. Außerdem akzeptiert AUSTRO-COMP EXTENSIONS (Befehle, die nicht im Basic 4.0 vorkommen), wie z.B. beim SCREEN-GENERATOR und führt diese auch aus.

Bei der Erstellung des AUSTRO-COMP wurde auf folgende fünf Punkte geachtet:

- volle Kompatibilität mit COMMODORE BASIC 4.0
- Schnelle Ausführungszeit
- Kurzer P-Code
- Möglichkeit von EXTENSIONS
- RUN-TIME Routinen maximal 4k Bytes lang

AUSTRO-COMP erzeugt von einem BASIC-Programm (Source-file) einen P-Code Mit einem speziellen Maschinenprogramm (RUN-TIME-Routinen), das ca. 4 k Bytes lang ist, dem P-Code und den im BASIC-programm enthaltenen Variablen wird ein kompiliertes ablauffähiges Programm (Objekt-file) erstellt.

Der schnellere Ablauf des mit AUSTRO-COMP kompilierten Programmes gegenüber dem Interpreter wird vor allem durch folgende Besonderheiten erzielt:

- GOTO und GOSUB-Sprünge werden direkt ausgeführt,
- Auf VARIABLE wird direkt zugegriffen

- Zahlen werden beim Kompilieren in Floatpoint oder Integer umgewandelt
- SYNTAX-Prüfung entfällt, da schon beim Kompilieren geprüft,
- Integer-Arithmetik ist vorhanden Schnellere Rechenzeit !
- Optimierung der Rechenfunktionen (keine Klammern !!)

Der von Austro-Comp erzeugte P-Code hat eine Länge von ca. 60% eines dicht geschrieben Basic-Programms (ohne Space und REM). Da jedoch die RUN-TIME Routinen mit 16 Blocks 4k-Bytes) dazukommen, wird ein kompiliertes Programm erst bei einer Basic Programmlänge von mehr als 40 Blocks kürzer als das BASIC-Programm (Source-file).

Führt die Verlängerung eines Programmes durch die RUN-TIME Routinen zu einem OUT OF MEMORY Fehler, (Kurzes Programm aber viele Daten), gibt es den AUSTRO-COMP ZUSATZ. bei dem sich die RUN-TIME Routinen in einem ROM für den freien Sockel mit der Adresse:

HEX 9000-9fff

befinden.Das kompilierte Programm (Objekt-file) kann im Gegensatz zum Basic-Programm nur auf einem CBM mit BASIC 4.0 ablaufen.

5. Kompilieren

Kompilieren mit AUSTRO-COMP ist sehr einfach. Der Compiler wird in den Arbeitsspeicher des CBM-Computers geladen und mit RUN gestartet. Danach benötigt AUSTRO-COMP kein zusätzlichen Programme oder Sequenzen für sein Ausführung. Nach Beendigung des Kompilierens kann sofort mit RUN neu gestartet werden.

AUSTRO-COMP stellt vier Programmarten zur Auswahl:

- Ein Programm kompilieren nach Drive 1
- Alle Programme von Drive 0 auf leere Diskette in Drive 1 kompilieren (alle (alle Programme mit Comp-Maschinen-programm)
- Alle Programme von Drive 0 auf leere Diskette in Drive 1 kompiliern (nur erstes Programm mit Comp-Maschinen-programm)
- Alle Programme von Drive 0 auf leere Diskette in Drive 1 kompilieren (mit (mit OVERLAY)

Die Unterschiede der einzelnen Programm-arten werden im folgenden beschrieben.

Austro-Comp kompiliert in 2 Durchläufen (PASS), bei OVERLAY in 3 Durchläufen. Das Das bedeutet, daß die Kompilierung in 2 bzw. 3 Durchläufen (Passes) abläuft.

5.1 PASS 1

Der Pass 1 übernimmt folgende Aufgaben:
Das BASIC- Programm (source-file) wird in einen P-Code übersetzt und dabei auf folgende Fehler geprüft:

SYNTAX-ERROR und TYPE MISMATCH ERROR bei Erkennen eines dieser Fehler wird dieser Fehler und die dazugehörige Zeilennummer am Bildschirm gelistet. Dieser Befehl wird nicht übersetzt. Ein eventuell 2. Fehler bei diesem Befehl wird nicht mehr ausgegeben. Der Compiler unterbricht das Übersetzen nicht. um eventuell noch weitere Fehler zu finden. Das kompilierte Programm (Objekt-file) sollte aber nicht benutzt werden.

Weitere Fehlermeldungen. bei denen jedoch der Befehl übersetzt wird. sind:

BAD SUBSCRIPT ERROR IN (ZEILENNUMMER)

Er wird gemeldet. wenn einem ARRAY verschieden Dimensionen zugewiesen wurden. z.B.:

x$(4.5) und x$(2).

Da es jedoch möglich sein kann. durch CLR neu zu dimensionieren. soll dies nur als Hinweis betrachtet werden.

OVERFLOW ERROR IN (ZEILENNUMMER)

Diese Meldung erfolgt, wenn eine Zahl im Programm entdeckt wird, die größer als 1 E 38 ist. Es wird eine falsche Zahl übersetzt

Werden EXTENSIONS erkannt, werden diese gegelistet, außer es handelt sich um Spezialanweisungen (siehe Spezialanweisungen, Seite 19).

Im 1.Pass werden sämtliche Zeilennummern und Adressen auf die ein GOTO, GOSUB oder THEN zeigt, sowie alle Variablennamen und Arraynamen gespeichert. Im Gegensatz zum Interpreter, bei dem die Variablen in der Reihenfolge ihres Auftretens im Programmablauf gereiht sind. werden hier die Variablen in der Reihenfolge ihres Auftretens im Programm gereiht.

Am Ende des 1.Passes wird geprüft, ob alle mehrdimensionalen Arrays auch dimensioniert wurden. Ist dies nicht der Fall, kommt die Fehlermeldung:

BAD SUBSCRIPT ERROR VON (ARRAYNAMEN)

Bei eindimensionalen Arrays, die nicht dimensioniert wurden. wird beim Start und nach jedem CLR ein Array mit 11 Elementen (0-10) wie beim BASIC 4.0 erzeugt.

5.2 PASS 2

Im 2. PASS wird das Objekt-file zusammengestellt, Zuerst die RUN-TIME Routinen, danach die DATA's (soweit vorhanden) Als nächstes folgt der P-Code, in den in diesem Durchlauf die Sprungadressen eingesetzt werden. Ist eine Zeilennummer nicht vorhanden, erfolgt bei diesem Pass die Fehlermeldung:

UNDEF'D STATEMENT ERROR IN(ZEILENNUMMER)

Zuletzt wird die Variablentabelle angehängt.

6. Beschreibung der Programmarten

6.1 Ein Programm kompilieren nach Drive 1

Bei Wahl dieser Programmart wird nach dem Programmnamen des Basic-Programmes (Source-file) gefragt, welches kompiliert werden soll. Dieses Programm kann sich sowohl in Drive 0 als auch in Drive 1 befinden. Das kompilierte Programm wird in Drive 1 mit dem selben Namen und mit dem Vorsatz c/..... gespeichert.
Außerdem wird noch ein Zeilennummern-Fehlerprogramm mit dem Vorsatz z/..... erzeugt. Beim Kompilieren benötigt der AUSTRO-COMP noch bis zu zwei seq. Files mit dem Vorsatz p/..... für den P-Code und d/... für die DATA's, insofern solche auftreten.

BEISPIEL:
Basic-Programmname: test

```
c/test = Objekt-file
z/test = Zeilennummern-file
p/test = P-CODE
d/test = DATA-File
```

Die beiden seq. Files: p/... und d/... werden bei Übersetzungsende gelöscht. Verwenden Sie daher keine volle Diskette zum Compilieren, da sonst ein DISK-FULL Fehler auftreten kann.

Am Ende der Übersetzung wird die Anzahl der Fehler sowie die Anzahl der Extensions ausgegeben.

Am rechten oberen Bildschirmrand wird die Zeit, die für das Kompilieren benötigt wurde angezeigt. Danach werden sämtliche Files geschlossen und evtl. gelöscht. Der DISK-STATUS wird Invers ausgegeben. Das in Drive 1 befindliche kompilierte Programm c/"name" kann nun geladen und gestartet werden.

6.2 Alle Programme von Drive 0 nach Drive 1 kompilieren (alle Programme mit Comp-Maschinenprogramm

Bei dieser Programmart werden sämtliche Programmfiles von Drive 0 kompiliert und unter dem gleichen Namen auf Drive 1 gespeichert. Die Reihenfolge der Programme bleibt die gleiche wie die in Drive 0. Wird die Diskette in Drive 0 vom Programm aus formatiert, dann bekommt die formatierte Diskette den gleichen Header und die gleiche ID-Nummer wie die Originaldiskette im Drive 0. Im Gegensatz zu Programmart 1 wird kein Zeilennummernfehler-Programm erzeugt. Bei Auftreten eines RUN-TIME Fehlers (siehe Fehlermeldungen. Seite 25).
muß mit Programmart 1 ein z/File erstellt werden. Nachstehend folgt ein Beispiel:

Auf Diskette in Drive 0 befinden sich folgende Programme:

- file a
- file b
- file c
- file d

Nach dem Kompilieren befinden sich auf der Diskette in Drive 1 folgende Programme:

- file a
- file b
- file c
- file d

Wenn nun beim Ablauf des Programmes:

file c

ein RUN-TIME Fehler auftritt muß das BASIC-Programm

file c

mit Programmart 1 kompiliert werden. Danach kann mit dem neu erzeugten File

z/file c

die Basiczeilennummer, in der der Fehler auftrat, ermittelt werden. Siehe auch Fehlermeldungen. Seite 25.

Am Ende jeder Übersetzung wird die bisher benötigte Zeit am rechten oberen Bildschirm-rand angezeigt. Sonst wie Programmart 1.

6.3 Programmart 3
Alle Programme von Drive 0 nach Drive 1 compilieren. nur erstes Programm mit Comp-Maschinenprogramm
Diese Programmart soll gewählt werden, wenn es sich um ein Programmpaket handelt in dem das erste Programm als Menü-Programm dient, die Folgeprogramme aber unabhängige Teilprogramme sind. Dies bedeutet, daß in die Teilprogramme keine Variablen und deren Inhalt übernommen werden, diese vielmehr ihre eigenen Variablen verwalten.
Folgende Vorteile können damit erzielt werden:

1. Ladezeit der Folgeprogramme ist kürzer

2. Geringerer Platzbedarf auf Diskette

3. Nachfolgende Programme können nicht ohne Startprogramm gestartet werden, dadurch muß nur das erste Programm durch einen Programmschutz gesichert werden.

Im übrigen Ablauf wie bei Programmart 2

6.4 Programmart 3
Alle Programme von Drive 0 nach Drive 1 kompilieren (mit Overlay)
OVERLAY stößt bei kompilierten Programmen auf Schwierigkeiten, weil sich im Normalfall hinter jedem Programm die Variablentabelle befindet. Auch die Zeiger eines jeden Programmes sind verschieden.

AUSTRO-COMP macht mit einem eigenen Programm OVERLAY möglich ist. Folgende Regeln müssen jedoch beachtet werden:

1. Alle Arrays müssen spätestens in dem Programm, in dem sie vorkommen, dimensioniert sein. Geschieht dies nicht wird ein
BAD SUBSCRIPT ERROR
gemeldet und es muß neu kompiliert werden.

2. Nur an das erste Programm, das Start-Programm wird eine Variablentabelle mit allen Variablen aller Programme angehängt. Nur bei diesem Startprogramm wird bei Programmbeginn ein CLR erzeugt. Bei allen anderen Programmen wird beim Nachladen kein CLR erzeugt. Nur ein RESTORE und ein Rücksetzen des STACKs erfolgt wie beim normalen OVERLAY. Im Gegensatz zum BASIC-OVERLAY kann jedoch das Startprogramm nicht mehr nachgewerden, weil sonst sämtliche Variablen gelöscht werden.

Beispiel:
Folgende Programme sind vorhanden:

- start
- file a
- file b
- file c

Lädt das Programm "start" das Programm "file a" nach werden alle Variablen übernommen. Lädt nun das Programm "file a" das Programm "file b" oder das Programm "file c" nach oder umgekehrt, werden ebenfalls die Variablen übernommen. Wird jedoch von den Programmen "file a" oder "file b" oder "file c" aus das Programm "start" geladen werden alle Variablen gelöscht.

Nach Ablauf der ersten beiden Durchläufe. die fast gleich ablaufen wie bei Programmart 2 (die abgespeicherten Programme sind ohne RUN-TIME Routinen und ohne Variablentabelle), wurde die Länge des längsten Programmes gespeichert
Beim PASS 0 werden alle Variablen und Arrays in einer Tabelle gespeichert, Danach wird geprüft. ob dieses Programm auch das längste ist. Trifft dies nicht zu wird es mit Füllzeichen verlängert.
Im Anschluss die Variablentabelle aller Variablen aller Programme angehängt.
Bei den folgenden Programmen werden noch die Zeiger berichtigt und gespeichert.

7. Spezialanweisungen

Im Gegensatz zu Basic-Programmen ist bei kompilierten Programmen die STOP-TASTE zum Abbrechen des Programmes außer Funktion gesetzt.
Ebenfalls gesperrt ist bei einem kompilierten Programm die Möglichkeit, bei INPUT ohne Eingabe, durch Drücken der RETURN-TASTE das Programm zu unterbrechen.
Auch ein CONTINUE ist (durch den Unterschied zwischen Basic-Interpreter und RUN-TIME Routinen) nicht möglich. Statt CONTINUE ist SYS 1055 zu verwenden.

Um jedoch die STOP-TASTE oder die INPUT-Funktion in den ursprünglichen Zustand zu bringen (einschalten), sind folgende Spezialanweisungen möglich, die sich im Basic Programm (Source-file) befinden müssen und erst ab deren Auftreten gelten:

```
rem ** se          = STOP EIN
rem ** sa          = STOP AUS (wie nach Start)
rem ** ie          = INPUT EIN
rem ** ia          = INPUT AUS (wie nach Start)
rem ** sp          = DONGLE PRÜFEN
      donglenummer
rem ** e           = keine Extensions-Listung
                     waehrend des Kompilierens
::                 = EXTENSION
```

Bei leerem INPUT werden STOP und BREAK bei:
- jedem Start
- Nachladen und
- nach einem CLR

ausgeschaltet.

Die Dongleprüfung erfolgt an der Stelle, an der im Basicprogramm die Anweisung
- rem ** sp

gesetzt wurde. Diese kann an jeder beliebigen Stelle im Programm gegeben werden.

Wird ein EXTENSION vom Compiler als solche nicht erkannt, in diesem Fall meldete der Compiler:
?SYNTAX ERROR
kann durch zwei weitere "::" vor der EXTENTION diese als solche gekennzeichnet werden. In diesem Falle wird der gesamte Ausdruck zwischen "::" und dem folgenden " :: " übernommen.

8. Arithmetik und Funktionen

Der AUSTRO-COMP hat eine eigene Integer-Arithmetik. Er setzt sie immer dann ein, wenn es sich um echte Integer Zahlen handelt. Bei jeder realen Zahl wird geprüft, ob es sich um eine Integer Zahl handelt. Hierdurch kann Rechenzeit gespart werden. Beim Ergebnis gibt es keine Abweichung zum Basic-Interpreter.

Funktionen:
Alle Funktionen ergeben die gleichen Resultate wie die des BASIC-Interpreters. mit folgender Ausnahme:

Bei den INTEGER-FUNKTIONEN:

- PEEK (x)
- INT (x).

sowie bei den Befehlen:

- POKE (x.y)
- WAIT (x.y)
- SYS (x)

und bei der ARRAY - Berechnung wenn es sich um reale errechnete Zahlen handelt, diese Zahlen werden vor Umwandlung in eine Integerzahl beim Basic-Interpreter nicht gerundet. COMMODORE Basic 4.0 rechnet bei realen Zahlen mit einem 8 Bit Exponenten und einer 40 stelligen Mantisse. Abgespeichert und für die Integerumwandlung verwendet werden nur die ersten 32 Bit der Mantisse. Beim Abspeichern wird das 33. Bit abgefragt. Ist es gesetzt. wird der Wert um 1 erhöht. Bei der Umwandlung in eine Integerzahl wurde auf die Abfrage des 33. Bit verzichtet (vergessen?).

Dadurch kommt es zu folgende Ergebnissen

```
PRINT PEEK (145)
              listet den Speicherplatz 145
PRINT PEEK (145/3*3)
              jedoch listet den Speicher-
              platz 144
a=145/3*3: PRINT PEEK (a)
              listet jedoch wieder den
              Speicherplatz 145
```

AUSTRO-COMP rundet (nach reiflicher Überlegung, da nicht ganz kompatibel) reale Zahlen vor einer Umwandlung in eine Integerzahl.

9. Arrays

Sämtliche Arrays können dynamisch dimensioniert werden. Bei einem DIM-Befehl kann der Elementwert durch eine Variable ausgedrückt werden. z.B:

```
DIM x(l,r)
```

Eindimensionale Arrays mit maximal 11 Elemerten (0-10) müssen nicht dimensioniert werden. Da es jedoch bei Mischen von automatischer Dimensionierung und Dimensionierung mit DIM in Zusammenhang mit CLR zu Schwierigkeiten kommen kann, empfehlen wir, stets mit DIM Dimensionierungen vorzunehmen.
Sind im Source-file Arrays vorhanden dann erzeugt das kompilierte Programm beim Programmstart einen ZEIGERARRAY. Dieses Array ist ein Integerarray mit dem im Basic unmöglichen Namen von z*%. Dieses Array dient zur raschen Auffindung eines Arrays während des Programmablaufes und benötigt außer dem Header von 5 Byte noch je 2 Byte für jeden im Programm vorkommenden Arraynamen.

10. Stacktiefe

AUSTRO-COMP benötigt weniger Stackplätze als der Interpreter, dadurch können komplexere Programme erstellt werden, die mit dem Interpreter nicht ablaufen. Zum Vergleich
seien hier ein paar Beispiele genannt:

AUSTRO-COMP		CBM-Interpreter
GOSUB	3 Bytes	5 Bytes
FOR	16 Bytes	18 Bytes
FOR mit INTEGER	9 Byte	nicht möglich

FOR-NEXT Schleifen können beim AUSTRO-COMP auch mit INTEGER-Zahlen gebildet werden.
Diese Schleife ist rascher und benötigt weniger Stackplatz als eine FOR ... NEXT - Schleife mit realen Variablen

11. Fehlermeldungen

Der AUSTRO-COMP kann während der Übersetzung nur folgende drei Fehler einwandfrei erkennen:

- SYNTAX ERROR
- TYPE MISMATCH ERROR
- UNDEF'D STATEMENT ERROR

Alle anderen Fehlerarten sind logische Fehler, die erst beim Programmablauf erkannt werden können, sog. RUN-TIME-Fehler.
Da es im kompilierten Programm keine Zeilennummern gibt wird bei Erkennen eines Fehlers der Stand des Programmzählers als Adresse ausgegeben.
Im Zeilennummern-Fehlerprogramm z/file wird für jede Zeilennummer des BASIC-Programmes der dazugehörige Programmzählerstand des kompilierten Programmes festgehalten.
Wird nun beim Ablauf eines kompilierten Programmmes ein Fehler gemeldet wird, muß nur das dazugehörige z/file ohne RUN geladen werden
Danach erfolgt die Eingabe von LIST -Zeilennummer
Die als letzte gelistete Zeile zeigt nach dem " = " - Zeichen die BASIC-Zeilennummer an, in der der Fehler festgestellt wurde.

Beispiel:

```
   Programm: c/test
   Fehlermeldung: ILLEGAL QUANTITY
                  ERROR IN 5912
   Laden des Programmes: z/test
                         LIST -5912
           Letzte Zeile: 5905 = 520
```

In der BASIC Zeile Nummer 520 des Programmes "c/test" befindet sich der Fehler.

Um bei langen Programmen die Listzeit zu verkürzen, kann auch mit folgender Eingabe eingeengt werden:

LIST 5880-5912

Sollte jedoch keine Fehlerzeile gelistet werden so muß mit einem neuen LIST der erste Wert 5880 niedriger angesetzt werden.

11. Maschinenprogramm

Die RUN-Time Routinen des AUSTRO-COMP benutzen nur die Speicherplätze, die auch der BASIC-Interpreter benutzt. Der erste und zweite Kassettenpuffer werden außer (wie bei BASIC 4.0) für Disk-Befehle nicht verwendet. Auch ein an ein BASIC-Programm angehängtes Maschinenprogramm wird voll übernommen, weil der AUSTRO-COMP nach Erreichen des BASIC-Endes alle folgenden Bytes unverändert dem P-Code übergibt.
ACHTUNG: Einsprungadressen verändern sich, weil sich die Länge der Programme ändert.
Einsprungadressen können jedoch, wie beim BASIC-Programm dynamisch mit Hilfe der Variablen-beginn-Zeiger Speicherplätze 42 und 43(*256) (Programmende) errechnet werden, da der Einsprungsabstand zum Programmende gleich bleibt.
Im Maschinenprogramm dürfen sich keine direkten Sprünge befinden, weil diese dann ebenfalls nicht stimmen.
Auch das Nachladen eines Maschinenprogrammes ist möglich. So z.B. den SCREEN-GENERATOR. Es darf jedoch im Gegensatz zum BASIC-Interpreter keine Variable als Flag für das Laden verwendet werden, sondern man muß mit PEEK einen oder mehrere Speicherplätze prüfen, ob sich das Maschinenprogramm im Speicher befindet.

Beispiel:

BASIC-Nachlademöglichkeit (für kompilierte Programme nicht zulässig):

```
10 IF X = 0 THEN X = 1: DLOAD "screen generator"
```

Dies würde bei kompilierten Programmen eine Endlos-Schleife werden, weil nach dem Laden des SCREEN GENERATORS das Programm wieder bei der BASIC-Zeile 1 (sys 1052) beginnt und mit einem CLR X auf X auf Null gesetzt wird.
Folgende Möglichkeit funktioniert mit dem kompilierten Programm, als auch mit dem BASIC-Interpreter:

```
10 IF PEEK(32000)  124 OR PEEK(31700)  60
   THEN DLOAD "screen generator"
```

Die in 32000 und 31700 befindlichen Zahlen müssen vorher festgestellt und dann in der BASIC-Zeile eingegeben werden.
Der AUSTRO-COMP übernimmt beim kompilieren, nach einem SYS-Befehl bis zum " : " oder Zeilenende alles unverändert.
Da einige Programme dies als Anweisung für das Maschinenprogramm benutzen. Diese Maschinenprogramme müssen, wie beim BASIC-Programm, die Programmpointer 119/120 vor der Rückkehr auf den, den Daten folgenden " : " setzen.

12. SONDERANWEISUNGEN

Vom AUSTRO-COMP werden fünf Sonderanweisungen verwendet. Dies sind:

rem ** se	=	Stop ein
rem ** sa	=	Stop aus
rem ** ie	=	Input ein
rem ** ia	=	Input aus
rem ** sp dongle-nummer	=	Dongle prüfen
rem ** e	=	keine Extentionsliste
::	=	Extentionskennzeichen

CONTINUE

Das Prcgramm wird nach BREAK mit SYS 1055 fortgesetzt

STACKTIEFE

GOSUB	:	3 Bytes
For real	:	16 Bytes
For integer	:	9 Bytes

13. AUSTRO-COMP-DATEN

AUSTRO-COMP ist in BASIC geschrieben

SOURCE-FILE	:	73 Blöcke
P-CODE	:	40 Blöcke
RUN-TIME	:	16 Blöcke
VARIABLE	:	3 Blöcke
OBJEKT-FILE	:	59 Blöcke

Übersetzungszeit für dieses Beispiel:
6 min. 40 sec

GRENZWERTE:

BASIC-Zeilenlänge	:	255 Bytes
BASIC-Zeilenanzahl	:	1800 Zeilen je Programm
Sprungbedingungen	:	2500 je Programm (GOTO/ GOSUB/THEN)
Variablennamen	:	1000 Variable je Programm (außer OVERLAY)
Arrays	:	300 Arrays jeProgramm (außer OVERLAY
Array bei OVERLAY	:	255 Arrays

Stichwortverzeichnis Seite

Adressen 10
Array 10, 17, 21
Austro-Comp Zusatz 7
Bad Subscript Error 10, 11, 17
Besonderheiten 3
Break 20
CLR 17
Continue 19
Copyright 5
Data 11
DIM 23
Dongel prüfen 19
eindimensionales Array 11
Extension 3, 6, 10, 19
for .. next - Schleife 24
Funktionen 21
Geschwindigkeit 4
Gosub - Sprünge 6
Goto - Sprünge 6, 19
Input aus 19
Input aus 19
Integer-Arithmetik 7, 21
Kompatibilität 6
Kompilieren 8
Lizenz 5
Maschinenprogramme 3
Objekt-File 11
Out of Memory 7
Overflow Error 10
Overlay 17
P-Code 7, 9, 11
Pass 1 9
Pass 2 11

Stichwortverzeichnis Seite

Programmarten 8
Programmpakete 3
Programmschutz 4
Restore 17
Run-Time-Routinen 11
Stop aus 19
Stop ein 19
Syntax Error 9, 20
Syntax Prüfung 7
Type Mismatch Error 9
Undef'd Statement Error 11
Variable 6, 10
Variablentabelle 11, 17
Warm-Overlay 3
Zahlen 7
Zeilennummern 10

Alle Rechte, auch die der Übersetzung in fremde Sprachen, vorbehalten.
Kein Teil dieses Programms oder der technischen Beschreibung/der Spielanleitung
darf ohne schriftliche Genehmigung der Commodore GmbH
in irgendeiner Form reproduziert oder unter Verwendung elektronischer Systeme
verarbeitet, vervielfältigt oder verbreitet werden.

© Copyright by Commodore GmbH, Frankfurt 1983

Art.-Nr. 528032